Attorney Docket No.: 002489.P041 *PATENT*

UNITED STATES PATENT APPLICATION

For

MULTIPLE STRING SEARCHING USING CONTENT ADDRESSABLE MEMORY

INVENTOR:

Sunder Rathnavelu Raj

Prepared By:
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8300

"Express Mail" mailing label number: EV409362997US
Date of Deposit: 11/3/03
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" under 37 C.F.R. § 1.10 on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, PO Box 1450, Alexandria, Virginia 22313-1450.
JUANITA BRISCOE
(Typed or printed name of person mailing paper or fee)
(Signature of person mailing paper or fee)
11/3/03
(Date signed)

MULTIPLE STRING SEARCHING USING CONTENT ADDRESSABLE MEMORY

TECHNICAL FIELD

[0001] This invention relates to the field of string search devices and, in particular, to the use of a content addressable memory device to perform searches for multiple strings.

BACKGROUND

[0002] The problem of string searching occurs in many applications. The string search algorithm looks for a string called a "pattern" within a larger input string called the "text." Multiple string searching refers to searching for multiple such patterns in the text string without having to search multiple passes. In a string search, the text string is typically several thousand bits long with the smallest unit being one octet in size. The start of a pattern string within the text is typically not known. A search method that can search for patterns when the start of patterns within the argument text is not known in advance is known as unanchored searching. In an anchored search, the search algorithm is given the text along with information on the offsets for start of the strings.

[0003] A generalized multiple string search is utilized in many applications such as URL based switching, Web caching, XML parsing, text compression and decompression, analyzing DNA sequences in the study of genetics and intrusion detection systems for the internet. In string searching applications, an argument text is presented to the string search engine, which then searches this text for the occurrence of each of a multiple patterns residing

in a database, as illustrated in Figure 1. If a match is found, then an index or code that uniquely identifies the matching pattern entry in the database is returned along with a pointer (offset) to the matching position in the input text string. The pointer indicates the number of characters positions that are offset from the starting character of the string for which a matching pattern in the database is found in the input text string.

[0004] For example, consider the input text string: "We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness." Assume that the pattern "that" is stored in the pattern database as a first pattern (Pattern 1) and the pattern "are" is stored in the pattern database as a second pattern (Pattern 2). For the two pattern strings "that" and "are," a string search engine utilizing a matching algorithm may output a result of Offset-41/Pattern 1 because the pattern "that" was found as a pattern in the database and the first character "t" in the pattern "that" is offset 41 places from the starting character "W" of the input text string. The other results, for example, would be as follows: Offset-54/Pattern 2; Offset-73/Pattern 1; Offset 83/Pattern 2; Offset 145:/Pattern 1; Offset 162/Pattern 2.

[0005] Some prior string search engines are based on software algorithms such as Boyer-Moore that are inherently slow and have limited throughput. Other prior string search engines utilize the Aho-Corasick algorithm for string matching in which either a static random access memory (SRAM) or content

addressable memory (CAM) based lookup table is used to implement state transitions in the string search engine. One problem with prior string search engines utilizing the Aho-Corasick algorithm, such as disclosed in U.S. Patent 5,278,981, is that that they are incapable of performing wildcard or inexact matching. While some prior methods are capable of performing wildcard matching such as disclosed in U.S. Patent 5,452,451, the inexact matching feature is limited only to prefixes in text strings. Moreover, such prior methods are only capable of anchored searches in which the start of patterns within the incoming text string must be known and identified to the search engine. Further, such prior methods are not capable of case insensitive matching that is required in many applications. In addition, for a given pattern database, such prior methods require a large number of entries in a CAM device. In addition, the prior methods are not capable of increasing the search speed by processing multiple octets from the text string concurrently.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The present invention is illustrated by way of example and not intended to be limited by the figures of the accompanying drawings.

[0007] Figure 1 is a conceptual illustration of string searching.

[0008] Figure 2A illustrates one embodiment of a string search apparatus.

[0009] Figure 2B illustrates one embodiment of the string search apparatus of Figure 2A.

[0010] Figure 3A illustrates one embodiment of a ternary CAM.

[0011] Figure 3B illustrates one embodiment of fields of a ternary CAM and an associate memory.

[0012] Figure 4A is a state transition flowchart illustrating one embodiment of goto-failure method using an exemplary set of patterns.

[0013] Figure 4B illustrates an exemplary implementation of the goto-failure method of Figure 4A.

[0014] Figure 4C illustrates exemplary contents of one embodiment of a database having compressed entries implementing the goto-failure method of Figure 4A.

[0015] Figure 5 is a state transition flowchart illustrating one embodiment of a deterministic method for handling state transitions using the same exemplary set of patterns of Figure 4A.

[0016] Figure 6 illustrates an exemplary contents of one embodiment of a database implementing the deterministic method of state transitions of Figure 5.

[0017] Figure 7 is a flow chart illustrating one embodiment of a case insensitive search method.

[0018] Figure 7A shows the ASCII encoded character set.

[0019] Figure 7B shows one embodiment of a translation unit.

[0020] Figure 7C shows one embodiment of the character set after translation.

[0021] Figure 8A is a flow chart illustrating one embodiment of a method of wildcard matching.

[0022] Figure 8B illustrates one embodiment of a search string apparatus illustrating components implementing wildcard matching.

[0023] Figure 8C illustrates an embodiment of exemplary TCAM and associated memory fields implementing wildcard matching.

[0024] Figure 8D illustrates an alternative embodiment of a wildcard matching method with a fixed number of wildcard characters.

[0025] Figure 8E illustrates an alternative embodiment of a wildcard matching method capable of searching for nested wildcard patterns.

[0026] Figure 9A is a state diagram illustrating a parallel matching method using an exemplary set of patterns.

[0027] Figure 9B illustrates exemplary fields in an entry in a TCAM and exemplary registers in control circuitry.

[0028] Figure 9C illustrates an exemplary embodiment of TCAM and associated memory fields.

[0029] Figure 10A is a state diagram illustrating a rollback method for handling state transitions using the exemplary pattern set of Figure 9A.

[0030] Figure 10B illustrates entries that may be in a FIFO.

[0031] Figure 10C is a state diagram illustrating a rollback method for handling state transitions using the exemplary pattern set of Figure 9A.

[0032] Figure 10D illustrates an exemplary embodiment of TCAM and associated memory fields for a rollback matching method.

[0033] Figure 11 is a conceptual illustration showing a string matching apparatus handling multiple flows.

DETAILED DESCRIPTION

[0034] In the following description, numerous specific details are set forth such as examples of specific, components, circuits, processes, etc. in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that these specific details need not be employed to practice the present invention. In other instances, well known components or methods have not been described in detail in order to avoid unnecessarily obscuring the present invention.

[0035] Embodiments of the present invention include various method steps, which will be described below. The steps may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause hardware components (e.g., a processor, programming circuit)

programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software.

[0036] Embodiments of the present invention may be provided as a computer program product, or software, that may include a machine-readable medium having stored thereon instructions. The machine readable medium may be used to program a computer system (or other electronic devices) to generate articles (e.g., wafer masks) used to manufacture embodiments of the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnet or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions.

[0037] The machine readable medium may store data representing an integrated circuit design layout that includes embodiments of the present invention. The design layout for the integrated circuit die may be generated using various means, for examples, schematics, text files, gate-level netlists, hardware description languages, layout files, etc. The design layout may be converted into mask layers for fabrication of wafers containing one or more integrated circuit dies. The integrated circuit dies may then be assembled into packaged components. Design layout, mask layer generation, and the fabrication and packaging of integrated circuit dies are known in the art; accordingly, a detailed discussion is not provided.

[0038] It should be noted that the steps and operations discussed herein (e.g., the loading of registers) may be performed either synchronously or asynchronously. The term "coupled" as used herein means connected directly to or connected through one or more intervening components or circuits. Any of the signals provided over various buses described herein may be time multiplexed with other signals and provided over one or more common buses. Additionally, the interconnection between circuit elements or blocks may be shown as buses or as single signal lines. Each of the buses may alternatively be single signal lines, and each of the single signal lines may alternatively be buses. Additionally, the prefix symbol "/" or the suffix "B" attached to signal names indicates that the signal is an active low signal. Each of the active low signals may be changed to active high signals as generally known in the art.

[0039] A method and apparatus for text string matching is disclosed. In one embodiment, the method includes receiving a text string having a plurality of characters and using a state machine to perform a search on a database to locate instances of specific pattern strings in the text string. In one embodiment, the state machine includes a ternary CAM search engine. Performing the pattern search may include comparing a state and one of the plurality of characters in the text string with a current state and a current character, respectively, stored in the ternary CAM.

[0040] For one embodiment, the state machine looks for occurrence of one or more patterns stored in the database that match one or more characters in the

text. If a match is found, then an index that uniquely identifies the matching pattern in the database is returned along with an offset pointer to the matching position in the input text string. The pointer indicates the number of character positions that are offset from the starting character of the string for which a matching pattern in the database is found in the input text string. In one particular embodiment, the string matching apparatus may support the search of text string width's greater than the width of a row of CAM cells in the ternary CAM array.

[0041] In various embodiments, one or more of the following database search features may be supported: exact string matching, inexact string matching, single character wildcard matching (e.g., the pattern "Jo?n" where ? represents any single character, with such a pattern capable of matching incoming text string such as "John" "Joan" and 'Join" but not "Jon" or "Johan"), multiple character wildcard matching (e.g., the pattern "John had a # day" where # represents 0 or more characters, with such a pattern capable of matching an incoming text string such as "John had a good day" or "John had a AAABBB day"), case insensitive matching, parallel matching and rollback optimization, as discussed in further detail below.

[0042] Figure 2A illustrates one embodiment of a string search apparatus. String search apparatus 200 includes control circuitry 210 coupled to pattern and state database 215. Control circuitry 210 is configured to receive an input text string 205 having a plurality of characters from another device such as a

processor 100. (e.g., a network processor unit ("NPU"), microprocessor, or other control device including, for example, an Application Specific Integrated Circuit "ASIC" or the like). The control circuitry 210 is coupled to pattern and state database 215 to perform a search of the database for a stored pattern matching one or more characters of the input text string 205. Each character in the input text string may be encoded in one of the many encoding schemes known in the art, such as ASCII or EBSDIC. Typically, each character is encoded into one octet, although other encodings may be used. In one particular embodiment, the control circuitry 210 processes one character from the input text string at a time. Alternatively, control circuitry 210 may processes multiple characters at a time when a higher search rate is required. The multiple characters may be presented to control circuitry 210 at the same time or sequentially in time.

[0043] Figure 2B illustrates one particular embodiment of string search apparatus 200 of Figure 2A. In this embodiment, search string apparatus 200 includes control circuitry 210, search engine 220 and associated memory 230 that together operate as a state machine. Search engine 220 and associated memory 230 together form one embodiment of pattern and state database 215 of Figure 1.

[0044] Search engine 220 implements the string search function using a state transition scheme. The state transition information is collectively stored in the pattern and state database 215. Patterns are encoded in the search engine as a series of entries. In one embodiment, each entry in the search engine 220 is a concatenated word that includes one character of the pattern and the

corresponding state information. The control circuit 210 forms the search key (i.e., comparand) by concatenating one character from the input text with the current state information. The current state may be a null or idle state at power on. The control circuit 210 presents this concatenated search key to the search engine, which then searches through its entries. If there is a match, search engine 220 outputs a match index 225 that uniquely identifies the matching location in the search engine. If there are multiple matches, then the index corresponding to the highest priority is presented as index 225. Associated memory 230 receives the match index and returns data 235 stored therein. Associated memory 230 stores next state information and may store other information such as results and actions to be taken. When associated memory 230 returns the next state information, the next state information is written to the current state register or variable, and a new search may be performed of on the database stored in search engine 220. The above process repeats until an action is indicated by data 235 that halts the process. The control circuitry 210 may keep track of an offset that indicates the number of character positions that are offset from the starting character of the input text string 205 for which a matching pattern in the pattern and state database 215 is found and output the same to the processor 100 as results 250.

[0045] In one particular embodiment, search engine 220 implements the Aho-Corasick algorithm. Alternatively, the scheme described herein may also be

used to implement any large state machine involving a large number of states that may not be practical to implement by conventional means.

[0046] In one particular embodiment, associated memory 230 may be a random access memory (RAM) such as a static RAM (SRAM) or dynamic RAM (DRAM). In another embodiment, associated memory 230 may be a flash memory. Alternatively, another memory device, for example, a read only memory (ROM), such as an erasable programmable ROM (EPROM) or EEPROM may be used for memory 230.

[0047] In one embodiment, the search engine 220 comprises a ternary CAM (TCAM). Although discussed below in relation to a TCAM, in alternative embodiments, search engine 220 may be another type of search engine, for example, a hash based search engine or a trie based search engine. In one particular embodiment, a NSE5512 or NSE5526 ternary CAM available from NetLogic Microsystems, Inc. may be used for search engine 220. Alternatively, other search devices from NetLogic Microsystems, Inc. or from other vendors may be used.

[0048] Figure 3A illustrates one embodiment of a ternary CAM although other embodiments may be used. Ternary CAM 220 includes ternary CAM array 302, address decoder 304, priority encoder 306, flag logic 308, comparand register 310, instruction decoder 314, read/write circuit 312, and one or more global mask registers 311.

[0049] Ternary CAM array 302 includes rows of CAM cells for storing pattern data, and corresponding rows of mask cells for storing mask data. The ternary CAM array 302 can effectively store three states of information, namely: a logic one state, a logic zero state, and a "don't care" state for search or compare operations. The CAM array 302 cells may be any types of CAM cells including, for example, NAND and NOR based cells that may be formed from either volatile or non-volatile elements. Each CAM cell includes at least one memory storage element and at least one compare circuit. Other embodiments may be used to effectively implement an array 302 of CAM cells.

[0050] CAM words 0 to N-1 are each capable of storing a set of bits that may be received by comparand bus CBUS 326. CBUS 326 may be configured to receive search key 211 of Figure 2B. Data may be read from or written to TCAM array 302 over data bus DBUS 350 by read/write (R/W) circuit 312 that includes one or more sense amplifiers and one or more write drivers. Each CAM word 0 to N-1 is coupled to a match line 322_0 to 322_N, respectively. The match lines indicate whether comparand data matched data stored in CAM words 0 to N-1. Match lines 322_0 to 322_N are provided to flag logic 308 which generates a match flag signal /MF on line 334 indicating whether a match has occurred. Additional flags such as a multiple match flag may also be generated by flag logic 308. Flag logic 308 may also be incorporated into priority encoder 306. Match lines 322_0 to 322_N are also coupled to priority encoder 306. If one of the match lines indicates a match between the search key and data stored at a corresponding location in

TCAM array 302 (as masked by its local mask if set), priority encoder 306 outputs an index (e.g., an address) on RBUS 332 that uniquely identifies the location of the matching location in TCAM array 302. If more than one match is detected on match lines 322_0 to 322_N, priority encoder outputs the index associated with the highest priority entry in TCAM array 302. The highest priority entry may be located at the lowest physical address in TCAM array 302, at the highest physical address in TCAM array 302, or may use any other predetermined priority resolution scheme including operating on priority values explicitly stored with entries in TCAM array 302. Each CAM word 0 to N-1 has an associated local mask word 0 to N-1 that stores mask data for the CAM word. In contrast to global mask registers that mask entire columns of CAM cells, the local mask words include local mask cells 361 that mask individual CAM cells 363 of a corresponding CAM word on a bit-by-bit basis. The local mask cells may include memory cells for storing mask data. Each local mask word may include as many local mask cells 361 as there are corresponding CAM cells 363. For an alternative embodiment, there may be only as many local mask cells 361 as are required for masking corresponding CAM cells 363. For example, there may be less local mask cells 361 than CAM cells 363 if each of the CAM cell 363 will need not need to be masked. For alternative embodiments, the CAM words and local mask words may be encoded together to implement a ternary or quaternary function (storing effectively four states; namely, a 0, 1, always match or always mismatch state).

[0051] One or more global masking circuits (e.g., global mask 311) may be coupled between comparand register 310 and TCAM array 302 to mask entire columns in the TCAM array 302. It should be noted that TCAM 220 may include fewer components (e.g., comparand register may be omitted) or additional components than those shown in Figure 3A. As ternary CAMs are known in the art, a more detailed transistor level description is not provided.

[0052] Figure 3B illustrates one embodiment of fields that can be stored in one or more rows of TCAM cells of search engine 220, and one embodiment of fields that can be stored in one or more rows of memory cells in associated memory 230. In this embodiment, the TCAM fields include a state (STATE) field 351, a pattern character (CHAR) field 352, and the associated memory 230 fields include a next state (NXT_STATE) field 353, an action (ACTION) field 354, and a result (RSLT) field 355. The state field 351 and the character field 352 together identify a state transition. The size (e.g., the number of bits) allotted to fields 351 and 352 depends on the maximum number of states expected in the pattern and state database 215. The next state field 353 uniquely identifies the next state for a given comparand that matches a corresponding state and character in fields 351 and 352, respectively. The action field 354 contains an opcode that provides control information to control circuitry 210 indicating the action to be taken by string search apparatus 200. In one embodiment, for example, the action field may be 3 bit encoded with: a 000 value indicating no action, advance to next character (NOP); a 001 value indicating emit result stored in the result field and

advance to the next character in the input text string; and a 010 value indicating a failure with no advancement to the next character in the input text string. The size (e.g., the number of bits) allotted to field 354 depends on the maximum number of actions expected for the pattern and state database 215. The result field 355 contains a result code to be output from database 215 depending on the action. The size (e.g., the number of bits) allotted to field 355 depends on the maximum number of patterns in the pattern and state database 215.

[0053] In one particular embodiment, TCAM search engine 220 implements an Aho-Corasick (AC) algorithm. The AC algorithm uses finite state automata, also known as a state machine. Several methods for handling state transitions may be used when implementing the AC algorithm. In one embodiment, the method is a goto-failure method that achieves a reduction in the number of state transitions at the expense of lower throughput. In a given state, if any of the expected characters in any of the patterns is received, then the state machine goes to the next state. When the next character is not one of the expected characters, a failure link is taken to the state representing the longest prefix possible with the current state.

[0054] Goto-Failure Method

[0055] Figure 4A is a state transition flow chart illustrating the goto and failure method for handling state transitions using an exemplary set of patterns {he, she, his, hers}. A "goto" transition transitions to a new state while advancing to the next character in the input text. A "failure" transition advances

to a new state, but does not advance to the next character in the input text. Consider the state "she" 481. If the character "r" is received, the logical next state should be "her." However, the failure transition 461 jumps to state "he" 482 and once this state is reached, the character "r" 483 is considered again to make the correct state transition to "her" state 484.

[0056] The goto-failure method may be implemented using two tables to encode state-to-state transitions. The first table is a "goto" table that gives the next state value if a current character matches the expected character for this state. If there is no match in the first "goto" table, then the second table is used, which is a "failure" table that gives the state transition (a failure transition) if any other character is received. A failure transition may take the state back to the "idle" state in some cases. However, the next character can also take it to a state corresponding to a different pattern. Failure transitions reduce the throughput because the string search apparatus 200 advances to the next character only on a "goto" transition.

[0057] This goto-failure method may be implemented in TCAM search engine 220 and associated memory 230 by, for example, dividing TCAM search engine 220 into two blocks, as illustrated in Figure 4B. The states in the tables of Figure 4B may be identified with a unique descriptive string associated with the state for ease of discussion. In an actual implementation of the tables in TCAM search engine 220 and associated memory 230, each state is represented by a corresponding unique number.

[0058] All the goto transitions of the first table may be placed in a first goto block 491 with a higher priority (e.g., at a lower address). Each goto transition translates to one entry in the TCAM search engine 220 and one entry in associated memory 230. Within the goto block 491, the relative placement of the different transitions may not be important because only one of the entries in this block will match. All the failure transitions of the second table may be in a second block, failure block 492, following the first goto block 491. The relative position of the failure block means that its entries have a lower priority compared to the entries in goto block 491. The entries in the failure block 492 will match only if there was no match in the goto block 491.

[0059] In one embodiment, the goto-failure method may be optimized by compressing the entries in the blocks, as illustrated in Figure 4C below. In this embodiment, all failure transitions to the state IDLE (e.g., as shown by the four failure transition IDLE states 471-474 of Figure 4B) are captured by a single entry 475, for example, at the lowest priority entry of TCAM search engine 220 that has all the entries masked (represented by the * in the state field 351 and character field 352) and, therefore, will always result in a match.

[0060] The goto-failure method requires two look-ups for one incoming character in case the failure transition is taken, thereby resulting in reduced search speed. In an alternative embodiment, a deterministic method may be used that eliminates failure transitions. In this embodiment, state transitions

may be increased with the string search apparatus 200 making explicit transition from each state for each character.

[0061] Deterministic Method

[0062] Figure 5 is an exemplary state transition diagram illustrating a deterministic method for handling state transitions using the same exemplary set of patterns of Figure 4A. The deterministic method described below achieves a higher speed than the goto-failure method described above, but at the cost of extra transitions. In this embodiment, in each state, only one transition leads to the valid next state. This method is deterministic, since each character results in one lookup. The transitions shown in Figure 5 with the dashed lines are the new transitions over the state transitions of the goto-failure method shown in Figure 4A. In addition, for the sake of clarity, the transitions from any state to the "idle" state 510 and the transitions back to state "h" 486 and state "s" 487 are not shown. The deterministic implementation adds additional transitions shown with the dashed lines 451-455 to the goto block 491 of Figure 4B. It should be noted that not all transitions are shown for clarity. As an example consider the character "h" 586 is received in any state including the idle state 510, the state should transition to the state "h" 486 if "h" is not a regular transition. One such state transition 459 is marked with double line arrow going from state "he" 488 back to state "h" 486 upon receiving the character "h" 586. The rest of such transitions, although required, are not shown for clarity. A brute force implementation in one embodiment would have one TCAM search engine 220

entry (and associated memory entry 230) for each of the transitions. The implementation of such a brute force embodiment will end up with 31 entries for the example shown. The use of the ternary feature of TCAM search engine 220 lends itself to a very good compression of the entries. The entries can be reduced, for example, by dividing the entries in to three blocks as illustrated in Figure 6.

[0063] Figure 6 illustrates an exemplary structure of one embodiment of a pattern and state database implementing a deterministic method of state transitions. Pattern and state database 215 may be divided into three blocks: "block 1" 591, "block 2" 592 and "block 3" 593. These blocks correspond to the relative position of a state in the state transition diagram Figure 5. The block 593 with the lowest priority corresponds to the state "idle". This is the default entry that always goes back to idle state 510, if there are no other matches. In such an embodiment, all transitions to idle state 510 can be achieved with the single last entry of block 593. This entry will have all its fields masked (as indicated by the * in the state field 351 and the character field 352) and, hence, will always match resulting in a transition to the IDLE state 510.

[0064] All transitions corresponding to the states immediately following the "idle" state 510, such as the state "h"486 and state "s" 487, are implemented using block 592 containing entries with the next higher priority. These entries have the STATE field 351 masked out (as indicated by the * in this field). These entries will also take care of a transition from any state to the next state shown,

such as the transition 459 shown by the double line arrow. All other transitions go in the highest priority block 591.

[0065] Case Insensitive Matching

[0066] Figure 7 is a flow chart illustrating one embodiment of a case insensitive search method. In this embodiment, the method for handling state transitions accommodates case insensitive matching. As an example of a case insensitive match, the pattern "she" should match "she" or "SHE"." Alternatively, case insensitive matching may be required on certain portions of the pattern. As an example, "she" should match "She" but not "SHE" in a case where case insensitive matching is only used for "s" and "S". The case insensitive search method includes determining an encoding relationship between an upper case character and a lower case character at 710. Then, at 720, a comparison of the input text string 205 with patterns stored in pattern and state database 215 is performed that is independent of the case encoding relationship.

[0067] Figure 7A shows the American Standard Code for Information Interchange (ASCII) format encoding 730, which is one possible encoding for characters. In one embodiment, the characters of incoming text string 205 may be encoded in the seven bit ASCII format. A study of this format reveals that there is a fixed relation between the encoding of lower case and upper case characters. For example the lowercase character "a" is encoded in binary as 110 0001 (i.e., row 6=110 and column 1=0001). The upper case "A" is encoded as 100 0001. These two differ in bit position 5. This is true for all other alphabet characters as

well. If bit-5 can be masked out during a compare operation, case insensitive matching can be achieved. This rule applies to all the alphabetic characters. As already described, each position in a ternary CAM can be set to a "don't care". In order to achieve the case insensitive matching for the text and patterns in the ASCII encoding example, bit-5 can be locally set to a "don't care" in all the patterns in the database where case insensitive matching is desired. The case insensitive matching can also be achieved for all the patterns in the pattern and state database, for example, by setting a global mask such that bit-5 is masked. In other example, extensions to the ASCII set such as the 8-bit ISO8859 may also be used.

[0068] Using the seven bit ASCII character set and masking bit-5 may, however, have an undesired side effect with respect certain special characters such as "[" that are also encoded in rows-4 and 5 along with the alphanumeric characters. If case insensitive matching is desired globally and so global masks are used and special characters 731 are used as part of pattern database, then incorrect operation may result since a character such as "[" will match both the characters "[" as well as "{". An alternative embodiment, a translation unit may be used to translate the 7-bit incoming ASCII characters to 8-bit outgoing characters as shown in Figures 7B and 7C. The special characters now appear in other unused rows in an expanded 8-bit table. While using one extra bit, this scheme allows case insensitive matching without any constraints. This is made possible because of the extra code space that is available in an 8-bit space. The

translation scheme should be applied to all the patterns stored in the database as well as to the incoming text characters before they are used in any compare operations. The scheme shown in Fig 7C is exemplary and any similar translation scheme can be used to achieve the same end. For one embodiment, 7-bit to 8-bit translation can be performed by translation unit 715 that may be included within pattern and state database 215. Translation unit 715 can be, for example, a lookup table, combinatorial logic, and any form of software or hardware that performs the necessary translation.

[0069] Wildcard Matching

[0070] Figure 8A is a flow chart illustrating one embodiment of a method of performing wildcard matching using state and pattern database 215. In such an embodiment, a search may be performed for patterns matching an input text string 205 having one or more of the characters unspecified. When a wildcard match is performed, the input text string 205 containing the wildcard may be conceptually split into, for example, two sub-patterns. The first sub-pattern contains the portion of the input string preceding the wildcard, called the prefix. The second sub-pattern contains the portion of the input text string 205 succeeding the wildcard, called the suffix. Wildcard matching is used to look for any pattern matching the given prefix and the suffix of the input text string 205. As mentioned, the wildcard may comprise more than one unspecified character. In other words, there can be any number of intervening characters (including zero) between the prefix and the suffix. Consider, for example, the pattern

"T#BLE." "T" is the prefix, "BLE" is the suffix, and "#" represents the arbitrary number of unspecified intervening characters. The following patterns will match the above wildcard pattern: "TABLE," "TROUBLE," "TREMBLE," and "TUMBLE."

[0071] Figure 8A illustrates an exemplary flow diagram for wildcard matching. At 810, input information from the input string is searched against the stored patterns in the state and pattern database 215. At 820, a suffix is located and the process determines that a prefix corresponding to this suffix was previously found, a wildcard match has been located and a result indicating the match is output at 821. If, at 830 however, a prefix is found, then at 831 the result code corresponding to the prefix is output from the pattern and state database 215 and is stored (e.g., in the CUR_PREFIX 881 register shown in Figure 8B). . If, however, a non-wildcard match is found at 840, a result indicating this match is output at 841 and the process returns to 810. If no matches are located in the pattern and state database, the process performs 810 again with the next character from input text.

[0072] Figure 8B illustrates one embodiment of a string search apparatus that is capable of performing wildcard matches. In this embodiment, control circuitry 210 includes First-In-First-Out (FIFO) storage circuit 871, state registers 880, counter 891, clear logic 831, result logic 837 and register 815. For other embodiments, roll back circuitry 1070 may also be included.

[0073] FIFO storage circuit 871 is configured to receive characters of input text string 205, and outputs the characters to CUR_CHAR register 883 of state registers 880. In alternative embodiments, FIFO storage circuit 871 may be omitted and the input text string provided directly to CUR_CHAR register 883 or to a translation unit (e.g., translation unit 715 of Figure 7B).

[0074] State registers 880 include multiple registers containing various information used to perform a lookup in the ternary CAM array 302. For example, in the embodiment implementing wildcard matching, state registers 880 include current character (CUR_CHAR) register 883, a current state (CUR_STATE) register 884, a current prefix (CUR_PREFIX) register 881, and a count register 882. Alternatively, state registers 880 may be a single register having multiple register bit position groups corresponding to registers 881-884.

[0075] State registers 880 provide the search key for TCAM search engine 220. TCAM search engine 220 looks for the occurrence of one or more patterns stored in CAM array 302 that match the information in state registers 880. If a match is found then a search result is presented to associated memory 230 as a match index 225 corresponding to the matching location in the TCAM array 302. The match index 225 is used as the address 231 for a look-up of associated memory 230. Associated memory 230 stores additional data such as the next state, result, and action. An example of an entry in associated memory 230 is shown as entry 838. Associated memory 230 is coupled to control circuitry 210 to transmit the next state, action and result code data to the control circuitry 210.

[0076] Associated memory 230 may be coupled to register 815 of control circuitry 210. As discussed above in regards to Figure 8A, if a result is to be output at 821 and 841 in TCAM search engine 220, the result from the RESULT field of the corresponding entry in associated memory 230 is output for storage in register 815. For one embodiment, one or bits of the action field of a given entry in associated memory 230 can be used to control loading into register 815. This result may then be output from the apparatus 200 (e.g., to a processor such as processor 100).

[0077] The NXT_STATE field of entry 838 in associated memory 230 is coupled to current state register 884, such that the next state information corresponding to the match index 225 is loaded into current state register 884.

[0078] The action and result code data from entry 838 are coupled to result logic circuit 837 that loads the RESULT data from associated memory 230 into the CUR_PREFIX register 881 when a valid prefix result is encountered in a search of TCAM search engine 220.

[0079] The ACTION code is also provided to clear logic 831, for example, to assert a clear signal 832 that sets counter 891 to zero when a prefix in the text string 205 is detected after a search on TCAM search engine 220. For one embodiment, the action field may be 3 bits (A_2, A_1, A_0) encoded as follows: a 000 value indicating no action, advance to next character (NOP); a 001 value indicating emit result in the RESULT field; and a 010 value indicating a failure with no advancement to the next character. It should be noted again that the

action field of associated memory 230 illustrated in Figure 8B is only exemplary and the other action field codes/sizes and corresponding logic circuit configurations may be used.

[0080] Counter 891 is also coupled to receive an increment (INC) signal 833 that increments counter 891 for every new character received by control circuitry 210. The operation of count register 882 and counter 891 is discussed in more detail below in relation to Figure 8D. State registers 880 are also coupled to receive a power-on reset (RESET) signal 889 that loads an idle state in current state register 884.

[0081] It should also be noted that control circuitry 210 may not necessarily contain all the components illustrated in Figure 8B depending on what database search features may be supported by string search apparatus 200. For example, in an embodiment that does not implement wildcard searching, control circuitry 210 may not include clear logic 831, counter 891 and/or result logic 837. It should be also be noted that, alternatively, one or more of the component functions shown in the control circuitry of Figure 8B may be implemented within hardware or firmware of processor 100.

[0082] Consider the following example of the operation of apparatus 200 to locate a wildcard match in an input text string using Figures 8A-8C. Figure 8C illustrates an exemplary embodiment of TCAM search engine entries and associated memory entries that may be used in conjunction with the embodiment of control circuitry 210 shown in Figure 8B to store and search for the wildcard

pattern "T#BLE". Assume, for example, that string search apparatus 200 is in an idle state and receives a first character "T" from input string 205. The IDLE state is currently loaded in CUR_STATE register 884 and the "T" is loaded into CUR_CHAR register 883 and these contents are compared with the entries stored in TCAM search engine 220. A match is detected at address zero with the prefix "T", and the NXT_STATE of IDLE is read from a corresponding entry in associated memory 230 and loaded into CUR_STATE register 884. Additionally, the RESULT value of "101" and an ACTION value of "UPDATE CUR_PREFIX" are read from the corresponding entry in associated memory 230. In response to the action "UPDATE CUR_PREFIX", result logic 837 loads the RESULT value of "101" into CUR_PREFIX register 881. Now assume that one or more characters other than "B" are received from the input string text 205 and loaded into CUR_CHAR register 883. In each case, the TCAM search engine will be searched and no match will be found. When a "B" is received from input string 205, it is loaded into CUR_CHAR register 883 and the contents of registers 883, 884 and 881 ("IDLE", "B", and "101", respectively) are compared with fields 351, 352 and 856, respectively, in each of the entries stored in TCAM search engine 220. A match is detected at address one, and the NXT_STATE of "B" is read from a corresponding entry in associated memory 230 and loaded into CUR_STATE register 884. Additionally, the RESULT value of "0" and an ACTION value of "NOP" are read from the corresponding entry in associated memory 230. In response to the action "NOP", result logic 837 does not update the contents of

CUR_PREFIX register 881. If the next character received from input text string 205 is an "L", a match is detected at address two, the NXT_STATE of "BL" is loaded into CUR_STATE register 884, and the CUR_PREFIX register 881 is not updated. If the following character received is "E", a match is detected at address three, the NXT_STATE of "IDLE" is loaded into CUR_STATE register 884, the RESULT value of "102" and an ACTION value of "OUTPUT WILDCARD MATCH" are read from the corresponding entry in associated memory 230. In response to the action "OUTPUT WILDCARD MATCH", a wildcard match has been located because the suffix "BLE" was found and the suffix "T" was previously found as indicated by match between the value "101" stored in CUR_PREFIX register 881 and the value stored in field 856. The result 102 is loaded into register 815 and can be output from string search apparatus 200.

[0083] Figure 8D illustrates an alternative embodiment of a wildcard matching method with a fixed number of wildcard characters. In this embodiment, a fixed number of wildcard characters are searched for rather than an unbounded number of intervening characters in a wildcard match. As an example, consider the pattern "T??BLE" where each "?" represents a single wildcard character. "TUMBLE" will match the pattern while "TROUBLE" and "TABLE" will not match because of the incorrect number of intervening characters between the prefix "T" and the suffix "BLE". When the prefix is detected, in addition to storing the result in the previous result register 881, the control circuitry 210 maintains a count of the characters in the input text string

205 after a prefix match. This may be implemented, for example, using an internal counter 891. Internal counter 891 is set to zero when a prefix match is detected and, for every new character received, counter 891 is incremented by one. The count in counter 891 is also stored in COUNT register 882 and compared, along with the contents of registers 883, 884, and 881, with the entries in TCAM search engine 220, which also include a COUNT field 857. When a suffix pattern is detected, the values in the previous result field 856 as well as the count field 857 must match the corresponding values in the presented comparand in order for the wildcard pattern to be matched. As can be seen from Figure 8D, when the suffix "BLE" is detected (indicated by the address 3, current state 351 entry of "BL" and the current character 352 entry "E"), if the input text string 205 was "TUMBLE" then the count in address 3 count field 857 is 5, thereby resulting in a match because after "T" is detected there are exactly five characters received including the suffix characters "BLE". In the case of "TREMBLE," then there would be three characters "REM" between "T" and "BLE" generating a count of 6. Such a count of 6 will not result in a match.

[0084] Figure 8E illustrates an alternative embodiment of a wildcard matching method for identifying nested patterns. For example, assume two wildcard patterns "S#BLE" and "T#BLE", and an input text input string of "STABLE". "TABLE" is nested within "STABLE". As shown in Figure 8E, different result codes can be used to identify different prefixes (or suffixes) to accommodate nested wildcard patterns. For example, a first result code of "101"

can be used for identifying the detection of the prefix "S", a second result code of "102" can be used for identifying the detection of the prefix "T". Additionally, two different result codes can be used to identify when a first wildcard is detected and a second wildcard match is detected. For example, result code "103" can be used to identify when "T#BLE" is detected, and result code "104" can be used to identify when "S#BLE" is detected. In an alternative embodiment, the wildcard matching method can be enhanced to detect multiple nested wildcard patterns by having multiple CUR_PREFIX registers in the control circuitry and also having multiple PREV_RSLT fields in the TCAM search engine database 220. Additionally, the nested method can be extended for fixed number nested wildcard matching.

[0085] Parallel Matching

[0086] The methods described above are capable of very high speed searching. Figure 9A illustrates an embodiment of a parallel matching method capable of increased search speeds.

[0087] In one embodiment, the speed of the matching method may be increased by increasing the number of input characters that are compared at a time from the current one character to multiple characters. The one character at a time method considered so far achieves unanchored pattern matching. In going from one character at a time matching to multiple character matching, the main problem to be solved is how to achieve unanchored searching. This section describes how to achieve an N fold increase in search speed by considering N

characters from the input text at a time. Figure 9A illustrates an example of how to achieve 4X speedup by comparing 4 characters at a time. Consider the text "OPTICAL COMMUNICATIONS" and further consider that we are looking for the pattern "COMMUNICATIONS". When a set of 4 characters is presented to the string search apparatus, the start of the pattern within the text may be offset 0, 1, 2 or 3 characters within this four character group. In one embodiment, all four such possibilities are represented in the pattern and state database with the first, second, third and fourth state entries being offset by 0, 1, 2 and 3 characters, respectively. The string search apparatus 200 considers all four entries in the database, COMM 910, *COM 920, **CO 930 and ***C 940, in order to achieve an unanchored match (where the "*" denotes that the corresponding character in the database is masked out). Each of the states follows a separate branch path 901-904 through the state machine until the result state 950 is reached. By following the same search procedure for multiple patterns, the parallel matching method achieves un-anchored multiple string matching. The parallel matching method may be implemented in hardware by increasing the width of the state register 880, and correspondingly, increasing the width of the entries in TCAM search engine 220, by a size corresponding to a size of the number of input characters (N) that are desired to be compared at a time. For example, as illustrated in Figure 9B, if four characters will be processed at one time, then four CUR_CHAR registers 883_1-883_4 may be used and, correspondingly, four CHAR fields 352_1-352_4 may be used in each entry of TCAM search engine 220.

Figure 9C shows an exemplary implementation of the parallel matching scheme. The TCAM and associate memory space is divided into four blocks 960_1-960_4. The relative placement of the entries within a given block does not affect the operation. Block 960_4 is the lowest priority block (e.g., has the highest addresses) and contains the default entry to IDLE state. This entry will match when nothing else has matched. Block 960_3 is the next higher priority block and contains all transitions to the first level states after IDLE (i.e., states 910, 920, 930, and 940). All these entries have their STATE masked. This serves both the transitions from IDLE to the first level states as well as transitions from any other state to the first level states. Block 960_2 is the next higher priority block and contains all the entries where a result is output. The characters at the end that are not part of the pattern are masked out. The next state is the IDLE state. Block 960_1 is the highest priority block and contains all other entries. When a pattern is detected, the end of that pattern may take all four or only a part of the block of four characters. In this case, a possibility exists that the remaining characters may be the start of a new pattern. Hence, all the combinations of the end of the current pattern and start of all new patterns need to be included. For example, the entry at address 5 has a CHARS state of "ONSC". The first three characters "ONS" complete the current pattern and the result is output. The last character "C" may be start of a new pattern and hence the new state is "C".

[0088] The input text should be presented to the search apparatus in multiples of the set size (e.g., 4 characters as discussed above). When the last

set of characters in the text string are presented, it may not be equal to the full set size. In this case, the remainder of the characters in the set can be set to an unused character that does not occur in any of the patterns in the database.

[0089] Rollback

[0090] In Figures 9A-9C, a large number of entries in the TCAM search engine 220may be used for combinations of one pattern followed immediately by a second pattern. These entries can be eliminated and fewer entries needed in the TCAM search engine 220 through the use of a rollback method and apparatus described below. For one embodiment, FIFO storage circuit 871 can be used to store several incoming characters of the incoming text, and a read pointer of FIFO 871 can be used to selectively read out the desired characters stored in FIFO 871. A group of characters can be read from FIFO 871 and loaded into the corresponding CUR_CHAR registers. If, for example, the first two read characters match the end of a current pattern, the remaining characters can be effective ignored for this pass through the TCAM search engine 220. The remaining characters, however, remain stored in FIFO 871, and the read pointer of FIFO 871 can be rolled back or selectively set to point to access the remaining characters as part of a new search. Associated memory 230 may include one extra field per entry called the ROLLBACK field that identifies the number of characters that should be pushed or rolled back in FIFO 871. The rollback mechanism also allows further optimization by merging several branches of the state machine in to one.

[0091] Figure 10A is a state diagram illustrating a rollback method for handling state transitions using the exemplary patterns of Figure 9A. In this exemplary embodiment, for a given pattern, once the first N (e.g., four) characters are matched to a current state, then all the branches (e.g., branches 1001-1004) of the state machine converge to a single common lowest next state (e.g., state 1030) that is common to all the state transitions. In this process, if some of the current states have already progressed to more characters in the pattern than the others, these are then rolled back as shown in Figure 10A.

[0092] In the exemplary state diagram of Figure 10A, the pattern "COMMUNICATIONS" in the text string 205 is taken four characters at a time and exists as four branches of state transitions in the database 215 with the first, second, third and fourth branches being offset by 0, 1, 2 and 3 characters, respectively. The string search apparatus 200 considers all four possible entries COMM 910, *COM 920, **CO 930 and ***C 940. If the next four characters received in the input string 205 are "MUNI" 925, then the state machine transitions current state 920 to the *COMMUNI" next state 1030. If the next four characters received in the input string 205 are "UNI*" 915, then the state machine, which in the embodiment of Figure 9A would have gone to the state "COMMUNIC", instead rolls back the state to "COMMUNI" state 1030, even though the "COMMUNI*" state transition branch 1001 had progressed to more characters (e.g., 8 characters) than the "*COMMUNI" state transition branch 1002 (e.g., having 7 characters).

[0093] As another example, consider the state COMMUN 937. If the next four characters received in the input string 205 is "I***" 939, the state machine rolls back to "COMMUNI" state 1030 (a common state to another state transition branches) even though the "COMMUNI****" state transition branch 1003 had progressed to more characters (e.g., 10 characters) than the "COMMUNI" state transition branch 1004 (e.g., having 7 characters).

[0094] An embodiment of entries in FIFO 871 for the rollback method is shown in Figure 10B. The case shown in Figure 10B is from the state "COMM" 910 and when the input text "UNIC" 915 is received. Once these four characters are read, the read pointer points to the next valid character in the FIFO, which is character "A". Due to the rollback mechanism, the state diagram transitions to the state "COMMUNI" 1030, and the read pointer is rolled back to position 1021 to ensure that the next four characters read will be "CATI", and the input text 205 and the current state are in synchronization again. In one embodiment, using a circular buffer size of N, the write process stops writing when the FIFO count reaches N-3 to prevent overwriting the useful data that may be required in case a rollback takes place. The basic concept shown for a character-wide FIFO of Figure 10B can be extended to the parallel implementation for increased speed.

[0095] Figure 10C is a state diagram illustrating an alternative embodiment of rollback method for handling state transitions. In this embodiment, depending on the state, the rollback method processes some of the

input text string 205 characters twice. The read pointer is adjusted (rolled back) only when there is a partial match in the text with one of the patterns stored in database 215. The probability of a rollback can be reduced if the algorithm looks for a longer match before resorting to the rollback. Figure 10C illustrates an example where the string search apparatus 200 waits for a partial match of only 4 characters before starting the rollback.

[0096] Each entry in FIFO 871 may be wide enough (i.e., contain sufficient bits) to store one character at a time, or may be wide enough to store multiple characters at a time. For one example, each entry of FIFO 871 may be wide enough to store four characters in each entry.

[0097] In one embodiment, the rollback method discussed above with respect to Figures 10A-10C may be implemented in hardware by adding an extra field in the associated memory 230 and by adding rollback circuit 1070 (see Figure 8B) in control circuitry 210. Figure 10D illustrates an embodiment of the exemplary contents of a TCAM search engine 220 and associated memory 230 implementing a rollback method. This extra field is a ROLLBACK (ROLLBK) field that contains the count of characters that are rolled back in FIFO 871 before the start of a search.

[0098] Figure 10D shows an exemplary implementation of the rollback scheme described in Figures 10A-10C. The TCAM and associate memory space is divided into three blocks 1030_1-1030_3. Block 1030_3 is the lowest priority block (e.g., has the highest addresses) and contains the default entry to transition to the

IDLE state. Block 1030_2 is the next higher priority block and contains all the entries having as their next state (the CHARS state), the state after the IDLE state (i.e., states 910, 920, 930 and 940). Block 10301 is the highest priority block and contains all other entries. Looking at field entries 1042, 1043, 1044, and 1045, it can be seen that all four rows have the same next state. In effect, three of the input string entries are rolled back to match the stored pattern with the shortest match.

[0099] Figure 11 is a conceptual illustration showing a string search apparatus handling multiple flows or contexts. In many applications, there is a requirement to handle multiple contexts. In one embodiment, for example, the string search apparatus 200 may be used in a networking system to switch and/or route data between the network's transmission lines to transfer data payloads from a source to a destination. Data payloads are typically transmitted in the form of packets that are made up of smaller data cells. A packet is a unit of data that is routed between a source and a destination on a packet-switched network. Typically, a packet may travel through a number of network points having routers before arriving at its destination. When a data packet 1110 arrives at the input of router 1100, several lookups may be performed to determine the subsequent handling of packet 1110. Router 1100 may include processor 100 and a string search apparatus 200 to perform the various packet forwarding lookups. The packet 1110 may be parsed by processor 100 to get one or more keys (e.g., a header) in order to perform the various lookups.

[00100] Consider a typical Internet router employing the IPV4 based protocol system where multiple TCP/IP based connections exist. A single higher layer data payload may be split across multiple TCP packets. Multiple TCP connections may exist. Each TCP connection may generate multiple TCP packets. The TCP packets from different connections may be interleaved. Hence, when the TCP packets arrive at the string search apparatus, one choice is to re-assemble packets that belong to each TCP connection separately so that the text for a given connection is presented contiguously to the string search apparatus. However this method requires extra memory and also involves other overhead such as memory and protocol processing. An alternative embodiment considers each TCP connection as a separate context. When all the characters of a packet have been processed, it stores the current state or context of this connection. When a packet belonging to the same connection is received, then it restores the context and re-starts the search. In order to search through a higher-level data payload, the string search apparatus 200 switches between multiple contexts. This can be implemented as a simple table lookup (e.g., in memory 1120) to first fetch the context of the search. The context may include the parameters such as current prefix, result code, remainder of characters that could not be processed from the current packet, roll back value and count as discussed above. In case of parallel searching, for the example shown, a set of four characters is presented to the search apparatus. In case a packet is not a whole multiple of 4 characters long, a remainder number of characters, which may be up to 3 characters, may be

left. These characters are saved as part of the context and combined with the next packet belonging to the same TCP connection. The mechanism of saving and restoring the context allows the string search apparatus to handle multiple streams of input text that are interleaved in packets.

[00101] It should be noted that the string matching methods and apparatus discussed herein may be used in a wide variety of applications, for examples, URL based switching, Web caching, XML parsing, intrusion detection systems, implementation of data compression (e.g., Limpel-Ziv), calculations of DNA sequencing, and others.

[00102] It should be noted that the circuitry associated with the operations of a particular "diagram block" as illustrated within a "diagram block" is done only for ease of discussion and that such circuitry may have other groupings or physical locations within an apparatus.

[00103] In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense.